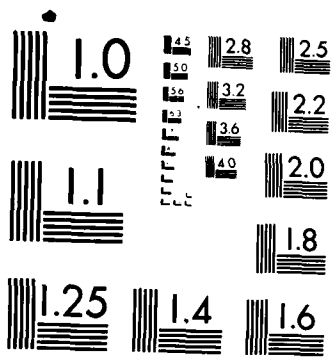
AD-A162 095 RESEARCH ON PROBLEM-SOLVING SYSTEMS(U) SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER D E WILKINS 22 OCT 85 AFOSR-TR-85-1091 1/1

UNCLASSIFIED F49620-85-K-0001 F/G 12/2 NL

END

FILMED

DTIC



(2)

RESEARCH ON PROBLEM-SOLVING SYSTEMS

INTERIM TECHNICAL REPORT
Period Covering: Oct. 1, 1984 to Sept. 30, 1985

October 22, 1985

By: David E. Wilkins
Representation and Reasoning Group

Artificial Intelligence Center
Computer Science and Technology Division

Prepared For:

Air Force Office of Scientific Research
Building 410
Bolling Air Force Base

Washington, D.C. 20332
Attention: Dr. Robert Buchal

F49620-85-K-0001

AFOSR Contract No. ~~F49620-79-C-0182~~
Project 7898

APPROVED

Michael P. Georgeff, Program Director
Representation and Reasoning Group

Stanley J. Rosenschein, Director
Artificial Intelligence Center

SRI International
333 Ravenswood Ave • Menlo Park, CA 94025
415 326-6200 • TWX 910-373-2046 • Telex 334-486

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR- 85-1091	2. GOVT ACCESSION NO.: AD-A162 095
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): RESEARCH ON PROBLEM-SOLVING SYSTEMS	5. TYPE OF REPORT & PERIOD COVERED: INTERIM TECHNICAL 10/1/1984 to 9/30/1985
	6. PERFORMING ORG. REPORT NUMBER: SRI PROJECT 7898
7. AUTHOR(s): David E. Wilkins, Artificial Intelligence Center, Computer Science and Technology Division	8. CONTRACT OR GRANT NUMBER(s): AFOSR Contract No. F49620-85-K-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS: SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304 A7
11. CONTROLLING OFFICE NAME AND ADDRESS: AFOSR/NM, Bldg 410, Bolling AFB DC 20332-6448	12. REPORT DATE: October 22, 1985
	13. NUMBER OF PAGES: 24
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See paper

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1 Introduction

Research on planning and problem-solving systems was begun at SRI International in September 1979 under AFOSR sponsorship (SRI Project 8871; Contract No. F49620-79-C-0188). The current project (SRI Project 7898; Contract No. F49620-85-K-0001), which started one year ago, continues this work. The research performed during previous years of the project is described in papers that appeared in the April 1984 issue of the Artificial Intelligence Journal [10] and the February 1985 issue of the Computational Intelligence Journal [11], as well as in five previous annual reports. The present report describes the research conducted during the past year.

~~Our main task under~~ this program ~~is to~~ develop powerful methods of representing, generating, and executing hierarchical plans that contain parallel actions. Execution involves monitoring the state of the world and, possibly, replanning if things do not proceed as expected. Over the last few years ~~we have~~ designed and implemented a system, called SIPE (System for Interactive Planning and Execution Monitoring) [10], the purpose of which is to demonstrate the heuristic adequacy of our approach to this problem. ~~Our~~ basic approach is to work within the hierarchical-planning methodology, representing plans in procedural networks - as has been done in NOAH [7] and other systems. Several extensions of previous planning systems have been implemented, including the development of a perspicuous formalism for describing operators and objects, the use of constraints for the partial description of objects, the creation of mechanisms that permit concurrent exploration of alternative plans, the incorporation of heuristics for reasoning about resources, and an implementation of a deductive capability.

~~Our~~ research this year has concentrated on interfacing the abstract, high-level plans produced by SIPE with the low-level information used by the sensors and effectors that

might interact with the real world (e.g., those installed on a robot vehicle). This has compelled us to add extensions to SIPE.

2 Interfacing SIPE with Low-Level Controllers

2.1 The Problem

The problem in using SIPE-like AI planners in a mobile robot environment is the gap that exists between the high-level reasoning of the planner and the lower-level control and signal processing needed by a robot. The planner deals with *conceptual* entities, while the lower-level routines deal with *perceptual* entities. The task we are undertaking is to map the perceptual entities of current lower-level systems to the conceptual entities of SIPE, and vice versa. We are using the planning of tasks for a mobile robot as a motivating domain. This has resulted in certain additions to the SIPE planning system.

We might view an integrated robot-control system as shown in Figure 1. Current robotic systems contain actuators and sensors, operating in conjunction with controllers that interpret their output, and generate commands according to a program that has been written specifically for the job at hand. Our research on SIPE during the last few years has concentrated on the planning system represented by the upper box in the figure. Little work has been done on the interface represented by the two arrows interconnecting the two larger boxes. There is a wide disparity in the representations and techniques used within each box; representations in the planning box are usually in some form of logic, while representations in the latter are often iconic or procedural [4]. Current perceptual reasoning is usually accomplished by special-purpose systems, such as free-space reasoning programs and vision systems. These problems cannot currently be dealt with by SIPE or, for that matter, any other high-level planning systems.

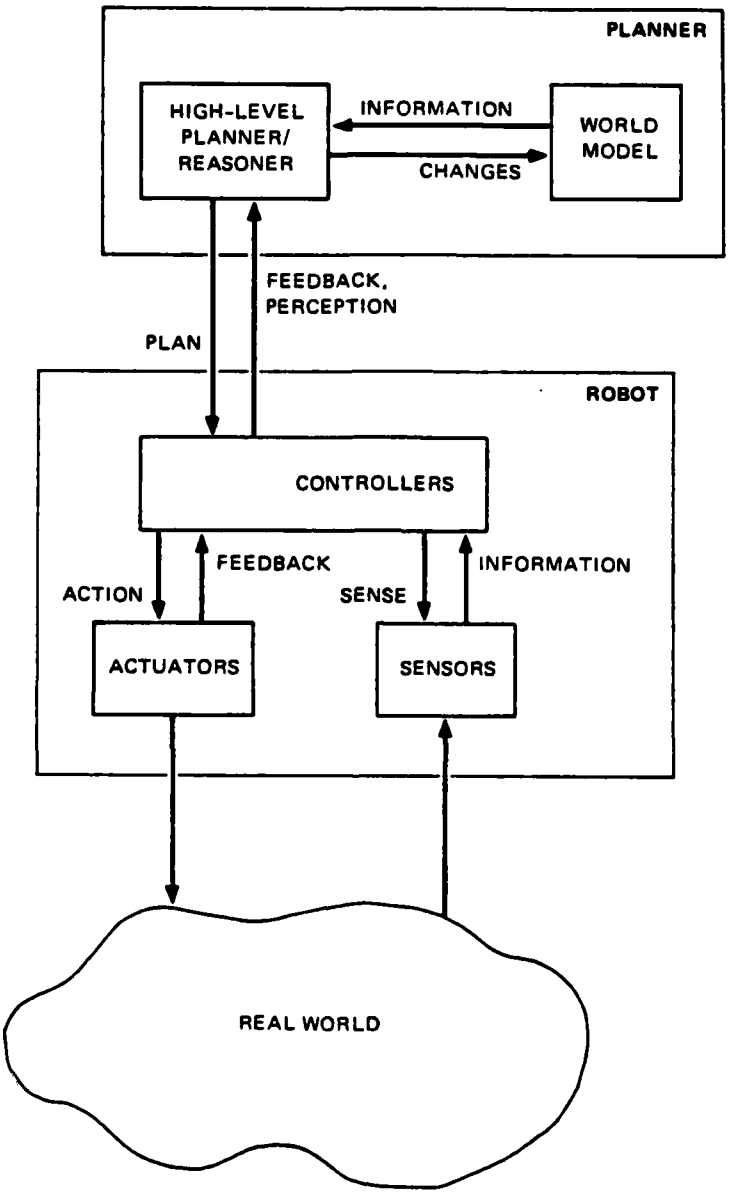


Figure 1: Structure of Robot Control and Planning System

Many lower-level control systems have been developed, each of which operates in a fairly specialized domain. These include various vision systems, systems for monitoring ultrasound sensors, programs that compute information from optical flow, programs that plan paths or reason about free space, and the like. They are specialized because they make assumptions about the domain. For example, model-based vision systems assume significant a priori knowledge about the geometry of the objects to be recognized [2]. Systems for analyzing optical flow may assume rigid body motion, or they may assume translation but no rotation in the movement of objects. Important gains in efficiency can often be obtained by enforcing such restrictions.

2.2 Our Design

Our design for interfacing SIPE with low-level controllers involves treating the controllers as programmable coroutines. There will be a two-way interface language for each controller that will allow SIPE to instruct and program the controller, while the latter, in turn, will be able to inform and instruct SIPE. The central idea is that the controllers will not be simply subroutines to be called (as in the limited amount of previous work that has been done on this problem), but will be routines running concurrently with SIPE that can send (as well as receive) information and requests. Thus, a sensor can monitor the world continually and alert the planner when a certain condition arises. The planner will be able to program such a sensor by telling it what conditions are expected and which unexpected conditions should generate interrupts. For example, when the robot is moving down a hall that is expected to be clear, SIPE wants to be alerted by the ultrasound sensor if an object is approached, but does not want to receive any messages if the space in front of the robot is clear. On the other hand, if the robot is moving up against a box to push it, the sensor's responses should be reversed.

Our approach allows the planner to instruct the sensors and effectors about what is expected and what should trigger an alert. It also provides for the sensors to request potentially useful information from the planner. One complication is that the instructions and requests that can be supported vary with each low-level controller. In general, a vision system will detect different situations than will an ultrasound sensor or speech understanding system. Thus, for each controller it will be necessary to develop an interface that provides for all the useful conditions the controller can recognize for the planner, along with all the types of information the controller can obtain from the planner and utilize in its processing. Furthermore, since the types of controllers needed are not available, either commercially or experimentally, controllers designed with this interface in mind will have to be developed.

The planner can help the controllers in many ways. Primarily, it will call these controllers in appropriate situations, given the domain restrictions imposed by the special-purpose algorithms. For example, it could call a specialized path planner only on paths that are expected to be clear. For other paths, the planner would plan to remove obstacles. There are various types of information that SIPE can provide these controllers to help them perform efficiently. On the basis of its knowledge of the world, the planner could manipulate some specialized internal representations of a controller. This might be complicated enough to require a special subsystem to generate or manipulate the special-purpose representations of the low-level systems according to predicates provided by SIPE. For example, the planner can provide a model-based vision system with a list of objects it expects to be in a scene, along with the expected camera angle between the robot and these objects. The vision system can then use this information to greatly reduce the search needed to interpret the scene.

This approach entails a rather loose coupling between the planner and the controllers.

This is in contrast to a more integrated system in which all aspects of the systems might be based on the same primitives. Rosenschein's use of situated automata is an interesting example of a more integrated approach [6]. The loose coupling is motivated by the desire to make maximum use of the many specialized algorithms that have been developed. SIPE operators would encode knowledge about when to apply these controllers, what their strengths and weaknesses are, where they can be used efficiently, and other relevant factors. The planner would then invoke a controller only in situations that match the particular constraints on its algorithm and with suitable a priori information. If SIPE produces reasonable plans, the controllers invoked should carry out their desired functions efficiently.

2.3 Controllers

The various controllers that are being considered for our mobile robot are listed below with a short description of each. Some of them are speculative, while others are certain to be included in the final system. At present, none of the systems mentioned below have been obtained, written, or interfaced with SIPE. Our research is still in the initial phases of overall system design.

- *Vision Verifier*

We have separated the function of verifying certain aspects of a scene from a general vision module that interprets or monitors arbitrary scenes. This has been done to take advantage of the fast techniques that have been developed for model-based vision. We plan to employ a vision system such as that described by Goad [2]. Such a system precompiles information about the objects it expects to see and then uses this for fast recognition. SIPE could help such a system by describing the objects it expects to be in the scene, their anticipated

relationships, and the predicted viewpoint of the camera. The results of applying the verifier could help SIPE recognize unexpected situations, as well as helping it to adjust its model of the locations of the robot and other objects.

- *Vision Monitor*

The monitor is a routine for recognizing objects in a more general or unexpected scene. As a rule, this may be very difficult and inefficient, but it is necessary when the robot finds itself in an unexpected situation. Many techniques could be used, depending on the domain. For example, analysis of shading may be quite useful in certain indoor domains [5]. While measurements of motion might be included here, they are mentioned specifically below in connection with optical flow.

- *Ultrasound Monitor/Verifier*

Ultrasound monitoring may be simple enough to allow both monitor and verifier to be considered as a single system. SIPE may plan to use this system to follow a wall along a hallway. The ultrasound may also discover hitherto unknown objects whose location can be determined; SIPE can utilize this information in planning to avoid such objects.

- *Path Planner*

There have been a number of routines written for path planning and free-space reasoning. Two notable examples are those of Brooks [1]and Thorpe [9]. SIPE should be able to interact readily with such systems. It can provide information about the objects that exist in the world, and can plan to traverse unobstructed routes. (This is described in more detail in the next section.) The problem eventually given to the path planner should, in most cases, be solved easily.

• *Bumper Monitor/Verifier*

This should be a simple routine. When the robot bumps into an unexpected object, the bumper monitor can give SIPE the location of one edge of that object. When SIPE wants to use the robot to push an object or locate it precisely, the robot can move until the bumper monitor indicates that it is in contact with the object.

• *Optical Flow Monitor and/or Verifier*

Measuring the motion of objects or the perceived motion of stationary objects while the robot is moving is very difficult. Specialized systems that operate in this domain generally make assumptions that impose severe restrictions on the domain to obtain some degree of efficiency [3]. For this reason, it is unlikely that such a controller will be used in our robot in the near future. Such a system might be useful for following along a hallway until a door appears, and would be necessary if the robot were to interact with moving objects. The latter does not seem feasible at present.

• *Teletype or Natural-Language Monitor and/or Verifier*

This is how SIPE currently accepts information about unexpected events, i.e., as predicates typed at a terminal. This will continue to be one means of providing the robot with information; in the long run, however, it may be replaced by a natural-language-understanding system, so that the robot can be instructed in English.

• *Arm Planner*

Eventually there will be an arm on the robot. There are many systems for controlling a robot arm, which is a key problem in industrial robotics. SIPE will rely almost entirely on such a special-purpose system for planning arm movements, merely giving it such higher-

level goals as grasping a specific object at a given location. SIPE will be responsible for getting the robot to a location from which the object can be grasped by the arm.

3 Applying SIPE to a Robot World

To evaluate SIPE's capabilities in the context of interfacing with low-level controllers, a substantial amount of effort was devoted to encoding a simple indoor robot world in SIPE's formalism and using the planning system to generate plans in this domain. The generation of reasonable and efficient plans required many extensions and improvements of SIPE. These are described in the next section. This section describes the domain we encoded, outlines the planning process effected, explicates the abstraction levels used, summarizes the predicates used to encode the domain, and describes various problems encountered.

We expended considerable effort this year on converting SIPE from Interlisp, running on a DEC-2060, to Zetalisp, running on the Symbolics 3600. While this change provides more computing power, its major justification is the superior programming environment on the Symbolics machine. This enables research effort to be translated into the program much more quickly and efficiently, making more productive use of the researcher's time.

3.1 The Problem and its Solution

The robot domain consists of some of the rooms in the Artificial Intelligence Center at SRI International. The robot, the rooms, and various objects were all measured to scale. The planning system used four levels of abstraction in the planning process, and produced a primitive plan that provided actual commands for controlling the robot's motors. The resulting plan was used to guide our robot simulator successfully through the task.

The robot world consisted of 5 rooms connected by a hallway. These were divided

into about 35 symbolic locations that were of interest to the planner. The world included multiple paths between locations, which greatly increased the amount of work done by the planner. The initial world was described by 222 predicates, about half of which were deduced from SIPE's deductive operators. The description of possible actions in SIPE included 25 operators describing action and 25 deductive operators.

The symbolic locations varied in size, but any place of interest could be one. They should be small enough so that the controller called to do the path planning can easily solve the problem of getting from one location to the next. For example, each doorway was a location, while long segments of the hall were also single locations. Each location had various attributes within SIPE, including the actual two-dimensional extent of the location, the coordinates of a focus, and the coordinates of a focus next to a door if one existed. From these data the planner can compute the symbolic location of any given real coordinate that might be returned by a sensor.

The problem given the planner was to get an object located in one corner of a distant room and deliver it to a desk in another room. The planning process required 7 levels of planning and took 35 seconds on a Symbolics 3600. About half this time was spent processing constraints that kept open (until the very end) the possibility of using either of two alternative paths to the critical locations. There was no backtracking. The planner produced 194 goal/process nodes and 70 additional control nodes in the plan; the most primitive level contained 58 goal/process nodes. 228 plan variables were created during the planning, most of them for deductions.

3.2 Levels of Abstraction

The operators use three different hierarchical levels below the task level. The tasks that can be described for the robot can employ as many hierarchical levels as desired above

these three, depending on the complexity of the tasks involved.

The first level below the task level is the planning of navigation from room to room. For example, if the task is to copy a paper for Phred, the room-level plan might be to go to the library to get the paper, then go to the copy center, and finally proceed to Phred's office. This does not involve any reasoning about particular doors or passages, though it may involve high-level predicates describing connections. These predicates would indicate that it is reasonable to move from one room to another, without bothering about any details as to how this might be done or whether it is possible in the current situation. When such a move is planned to a lower level, it may fail or many actions may have to be performed to clear a path.

Below the room level is the NEXT-TO level, which plans movements from one important object (that the robot is next to) to another. For example, to copy the paper, the robot will have to get next to the door of the copy center, then pass through the doorway, then get next to the desk of the operator, etc. This plans high-level movements within a room but is still not concerned with actual locations.

The lowest level is the location level, where SIPE plans movements down to the level of the actual locational grid it has been given. SIPE will call the path planner only when the situation indicates that there should be a clear path between two locations. Otherwise SIPE would plan to move some obstacle to clear the path or would plan to take a different path. SIPE will attempt to invoke the correct controller for the current situation. For example, an optical-flow system might be invoked if the robot is to move down a hallway until it sees a door. However, it is not practical to plan at a lower level than this in SIPE, as the system does not include a geometric representation.

3.3 Predicates Used

The predicates used to describe this domain are summarized briefly below. No claim is made that is ontology is optimal in any way - in fact, some predicates were chosen merely to test certain capabilities of the planner. The sort hierarchy includes (among other more obvious classes) BORDERs, which are boundaries between rooms that may or may not have doors at them, AREAs, which include rooms, halls, and lobbies, and LOCATIONs, which are the symbolic locations described above.

The following predicates never change value as actions are performed in the world:

(CONNECT BORDER AREA AREA)

These are given or deduced initially and, for each door, specify which two rooms it connects. Used to plan room paths.

(ADJACENT AREA AREA)

These are all deduced from CONNECT and are used for room path planning.

(ADJ LOCATION LOCATION)

These are given or deduced for all adjacent locations (most are deduced). Used for path planning at lowest level.

(MAILBOX HUMAN OBJECT)

These are all given initially and are used to accomplish DELIVER actions.

(ONPATH LOCATION LOCATION LOCATION)

This gives an intermediate location between the two outermost nonadjacent locations. Used for planning a path between the latter.

(DOOR-LOC BORDER AREA LOCATION)

All DOOR-LOCs are deduced. They tell which location in each room is next to the given

door and are used in planning to go through a door.

In contrast to the above group, the following predicates do change as actions are performed:

(AT OBJECT LOCATION)

Most ATs must be given except that every object that is ON something has AT deduced. AT specifies an object's location at the lowest abstraction level of symbolic locations. Because many deductions are made from it, it is the most important predicate at that level.

(ADJ-LOC OBJECT LOCATION)

These are given or deduced initially for all objects. They tell which locations an object is adjacent to and are used for getting into position next to objects. During planning these are deduced from ATs.

(INROOM OBJECT AREA)

Most initial INROOMS are deduced. INROOM specifies an object's location at the room level and helps solve problems of level coordination (see next section).

(CONTAINS LOCATION AREA)

Similar to INROOM, but is added dynamically because it is used to coordinate levels of abstraction (see next section). These predicates are posted as effects involving uninstantiated LOCATION variables, so that the room containing them can be known even though their instantiation is not. INROOM could have been used, but it is clearer to separate the different functions performed by CONTAINS.

(OPEN BORDER)

Given initially for borders and specifies whether they are clear.

(ON OBJECT SUPPORT)

These are given initially and trigger many deductions. They are used for noting that an object is on some support (which will determine its location and adjacencies).

(CLEAR LOCATION) (CLEAR BORDER)

All given initially, these specify whether a location is clear enough for the robot to navigate through it.

ON-MAIN-PATH, ON-ALTERNATIVE-PATH

These two predicates are not used in this robot application. The idea is for them to control room-level path planning where rooms can become impassable. In the current application, room navigation is accomplished by operators that look for two rooms connected by the same hallway.

3.4 Level Coordination

During this application of SIPE, a number of problems were addressed; two of the more important of these are discussed in this section and the next. The "level coordination problem" refers to the coordination of different levels of abstraction with different levels of the planning process. This is a problem that occurs in all hierarchical systems and is not unique to planners in the NOAH tradition (such as SIPE). The problem is that all necessary information at a higher abstraction level must be accessible before planning can be done at a lower abstraction level. An alternative is to produce the information on demand. But in either case, a hierarchical planning system must ensure that the amount of detail that will be required for reasoning in latter parts of the plan is present in its earlier parts.

The planning literature has confused what we will call ***abstraction levels*** with ***planning levels***. Abstraction levels, which are found in all hierarchical systems, involve a different level of abstraction in the predicates that are being reasoned about. In the robot world example, SIPE initially reasons about things at the room level, planning movement from

room to room without ever considering specific locations. This is a higher abstraction level than the one at which the path is planned from one specific location to the next. Planning levels occur in several planning systems; in particular, all those in the NOAH tradition (including SIPE). A new planning level is created by expanding each node in the plan with the operators that describe actions. This may or may not result in a new abstraction level, depending upon which operators are applied. In the blocks world described by Sacerdoti [7], there is only one abstraction level. All the hierarchical levels in the NOAH blocks world are simply planning levels.

In a planner like SIPE or NOAH, one part of the plan can attain a lower abstraction level than another part at the same planning level. This can cause problems because correct information may not be available at the lower abstraction level. This situation surfaced in the robot domain when there was a goal to get the robot AT a symbolic location later in the plan, but the early part of the plan was still working on INROOM goals and did not know the robot's location. There are many alternative solutions to this problem. For example, ABSTRIPS [8] solves it by assigning abstraction-level numbers to the predicates and planning a lower level only when appropriate. This could easily be implemented in SIPE. However, there are advantages in retaining the flexibility to plan certain parts of the plan to lower abstraction levels. Constraints generated during lower-level planning can narrow down the search, thus resulting in possibly large gains in efficiency. For this reason, SIPE allows the mixing of planning and abstraction levels.

Since SIPE allows this mixing of levels, the person writing the operators must ensure that this will not interfere with the application of operators. In the robot domain, only the planning of AT goals was affected when abstraction levels varied in the plan. This problem was solved by using the operator shown in Figure 2. It delays the solving of AT goals until the part of the plan preceding them has been brought to the same level of abstraction. It

```
OPERATOR: navto-notyet
ARGUMENTS: robot1,location2,area1,location1;
PURPOSE: (at robot1 location2);
PRECONDITION: (at robot1 location1),
     (inroom robot1 area1),
     (not (contains location1 area1));
PLOT: COPY
END PLOT END OPERATOR
```

Figure 2: Operator for Coordinating Levels

does this by checking whether the AT location of the robot is in the same room as the INROOM location of the robot. If the precondition of this operator fails, it means that the last AT predicate specified as an effect of an action preceded the last INROOM predicate specified as an effect. Consequently, the latter action must still be planned to the lower level of abstraction.

The navto-notyet operator is applied before any other to an AT goal (this is determined by the order in which operators are given). The plot of this operator is simply the token *COPY* that copies down the goal from the previous planning level. It was necessary to use a special token rather than specify the AT goal in normal syntax. Normally, SIPE inserts the precondition of an operator into the plan and maintains its truth during execution monitoring. In this case the precondition will not be true in the final plan, so the COPY option inserts the appropriate goal without also inserting the precondition of this operator. With this operator, SIPE can mix abstraction and planning levels freely without error in the robot domain.

3.5 Constraint Efficiency

Constraint efficiency is another problem that arose during the encoding of the robot domain. It is fairly easy to write operators that will produce constraints that are computationally too expensive. The person writing the operators must therefore be careful to formulate things in such a way as to ensure that the constraints will have a reasonable computation cost. This problem arose in the robot world during the deduction of the ADJ-LOC relationships. This predicate denotes all the locations that are adjacent to a given object. In SIPE, the deduction of this relationship after an object has moved involves both removing the old ADJ-LOCs that no longer hold and adding the new ones that have now become true.

The problem arose during removal of the old ADJ-LOCs that were no longer true. It seemed natural to write a deductive operator that deduced a negated ADJ-LOC for every location that was not adjacent to the location just moved to by the object. This works fine when the location being moved to is instantiated, but, when it is not, the number of things that can be not adjacent to it is enormous. A constraint is added to the variable in the negated ADJ-LOC effect in SIPE that must later be processed frequently during the matching required to determine the truth-value of predicates. The computation involved in processing this constraint slowed the system down unacceptably.

Fortunately, we were able to find an elegant solution in SIPE by making use of the universal variables and the NOT-SAME constraint supplied by the system. The removal of invalid ADJ-LOCs is now done by the same operator that deduces new ADJ-LOCs; the operator simply uses a universal variable that is NOT-SAME to the location variable that is now ADJ-LOC. In other words, if a location is not the same as an ADJ-LOC of the new location, then it is not ADJ-LOC to the new location. The deductive operator that deduces this is shown in Figure 3 as input to SIPE.

```
DEDUCTIVE.OPERATOR: updateadj
ARGUMENTS: object1,location1,
     location2 is not location1,
     location3 is not location1 class universal,
     location4 is not location3 class universal;
TRIGGER: (at object1 location1);
PRECONDITION: (at object1 location2),
     (adj location1 location3);
EFFECTS: (adj-loc object1 location3),
     (not (adj-loc object1 location4));
END DEDUCTIVE.OPERATOR
```

Figure 3: Operator for Coordinating Levels

4 Extensions of SIPE

A number of extensions were added to SIPE to enable the robot-world problem to be encoded efficiently. The most important addition was a complete redesign of the deductive capability to enable more powerful and useful deductions. This is described in the following subsection. Following that, we describe other changes, most of which are of such a technical nature that a proper understanding of them requires familiarity with planning systems. They are listed to illustrate the kind of detail that must be dealt with to produce an effective implementation. Much of the theoretical work done in this field simply ignores these practical issues.

4.1 Deduction in SIPE

In addition to operators describing actions, SIPE allows specification of deductive operators that deduce facts from the current world state. Domain-independent planners that have used a NOAH-like approach (as distinct from a theorem-proving approach) have not had a deductive capability. As more complex domains are represented, it becomes increasingly important to deduce the effects of actions from axioms about the world, rather than representing these effects explicitly in operators. Besides being necessary for execution monitoring, deduction is important for determining both side effects and conditional effects of an action.

SIPE maintains strict control over the application of deduction to prevent a combinatorial explosion, while still providing a powerful enough capability to be useful. All deductions that can be made are performed at the time an operator is expanded. The deduced effects are recorded in the procedural net, and the system can proceed just as if all the effects had been listed in the operator. Deductions are not attempted at other points

in the planning process. Deductive operators have triggers to control their application. (See Figure 3 for an example.) If the precondition of a deductive operator holds, its effects can be added to the world model (in the same context in which the precondition matched) without changing the existing plan.

SIPE previously allowed only one-step deductions and permitted universal variables when they occurred only in the effects of the operator. This year, however, we expanded these capabilities to allow multilevel deductions and the presence of universals in preconditions. Initially, deductions are triggered on the effects of a node. After all deductive operators have been applied, SIPE determines which newly deduced effects were not already true and applies the deductive operators to these recursively. This process continues until no effects are deduced that were not already true. This allows more powerful deductions while maintaining control of the deductive process.

The use of universals in preconditions involves a major change in the basic matching process of the planning system. The semantics of a universal in the precondition differs significantly from the use of universal quantifiers in formal logic. It does not mean that the system must prove that the precondition is true for all objects. It means rather that the system should generate constraints enabling the variable to match all and only those objects for which the precondition is true. This is a very useful and powerful capability. For example, in the deduction of ADJ-LOCs by the operator in Figure 3, the LOCATION3 universal variable has constraints that allow it to match only those locations that are adjacent to the object. This effectively picks out the subset of locations we are interested in and allows their efficient representation.

This use of universals involved a major change in the way SIPE determines the truth of a predicate. Before this extension, a universal variable matched any other variable in its class exactly. Thus, the matcher need look no further for possible matches of a predicate.

Allowing constraints on universals means that they no longer match exactly, but are rather only potential matches that depend on the instantiations of variables in the predicate being matched. The matching process must therefore continue collecting other potential matches and let the system generate additional constraints upon the possible ways the predicate could be made true.

SIPE includes a check for NOT-SAME universals, such as are generated by the operator in Figure 3. When looking for potential matches of a predicate, SIPE checks for two possible matches being negations of each other, with exactly the same arguments save one. If this one argument is a universal variable in both possibilities, and if the variables are constrained to be NOT-SAME to each other, then the system knows that this covers all possibilities and therefore ends its search for possible matches. The fact that this application of NOT-SAME universals is of general utility justifies inclusion of the foregoing check. This check for NOT-SAME universals illustrates the type of detail that an implementation must address in order to be efficient.

4.2 Other Extensions

This sections describes briefly some of the other extensions added to SIPE during the past year.

- Predicates can be specified as not changing while actions are being executed, thereby enhancing efficiency.
- Input syntax was improved, becoming more concise and readable.
- A new menu-based package was designed for controlling the planner interactively.
- Menu controllers were implemented for the whole system, allowing SIPE to be run interactively or automatically through menus. Menus also control the display of

information, the setting of parameters, and the execution monitor.

- A new method for encoding plan rationale was incorporated. This resulted in pervasive modifications of the execution monitor; problems can now be perceived and remedied more accurately than was possible hitherto.

- The matcher now puts more effort into checking whether NOTPRED constraints can be satisfied. This requires posting these constraints during the matching process, which means that they must be removed later if the match fails. More effort is spent in the matching process, but the system can find some failures immediately that would otherwise have been discovered only much later in the planning process.

- Parallel postconditions were correctly implemented in the planner, along with an ability to replan after the failure of parallel postconditions.

- The matcher now posts PRED constraints during the matching process. By posting constraints after the matching of the first predicate in a condition, the planner can shorten the search involved in matching the other predicates. Several complications had to be dealt with, however. The constraint must be removed if the match of a later predicate fails. The reason for not adding constraints originally is that the most concise constraint cannot be formulated until all predicates have been matched. Thus, the constraints added earlier might be large and could slow down all future matching. This was averted by implementing a check for constraint subsumption. The concise constraints are still computed and added after all predicates have been matched, but, if they subsume (or are subsumed by) any constraint already present, the subsumed constraint is removed.

- The resource allocation routine was made more efficient by incorporating the ability to improve the ordering of constraints for checking.

- Operators can now include the COPY option in their plot. This allows goals to be copied down without the insertion of preconditions (see Section 3.4).
- Constraint propagation in SIPE can now convert NOTPRED constraints to more efficient NOT-SAME constraints under appropriate circumstances.
- The whole system was made more robust, including more checks for errors and debuggings.

5 Publications

During the past year, a paper describing execution-monitoring and replanning capabilities within the SIPE planning system appeared in the Computational Intelligence Journal [11].

References

[1] Brooks, R., "A Subdivision Algorithm in Configuration Space for Findpath with Rotation", *Proceedings IJCAI-83*, Karlsruhe, Germany, 1983, pp. 799-806.

[2] Goad, C., "Fast 3D Model-Based Vision", in *From Pixels To Predicates: Recent Advances In Computational And Robotic Vision*, Editor, A. Pentland, Ablex.

[3] Hildreth, E.C., "Computations Underlying the Measurement of Visual Motion", *Artificial Intelligence 23*, 1984, pp. 309-354.

[4] Pentland, A.P., and Fischler, M.A., "A More Rational View of Logic or, Up Against the Wall, Logic Imperialists!", *AI Magazine*, Vol. 4, No. 4, pp. 15-18 (1983).

[5] Pentland, A.P., "Local Shape Analysis", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, March 1984, pp. 170-187.

[6] Rosenschein, S., "Formal Theories of Knowledge in AI and Robotics", Technical Note 362, SRI International Artificial Intelligence Center, Menlo Park, California, 1985.

[7] Sacerdoti, E., *A Structure for Plans and Behavior*, Elsevier, North-Holland, New York, 1977.

[8] Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces", *Artificial Intelligence 5 (2)*, 1974, pp. 115-135.

[9] Thorpe, C., "Path Relaxation: Path Planning for a Mobile Robot", *Proceedings AAAI-84*, Austin, Texas, 1984, pp. 318-321.

[10] Wilkins, D., "Domain-independent Planning: Representation and Plan Generation", *Artificial Intelligence 22*, April 1984, pp. 269-301.

[11] Wilkins, D., "Recovering from Execution Errors in SIPE", *Computation Intelligence 1*, February 1985, pp. 33-45.

END

FILMED

1-86

DTIC